Tcl8.6.14/Tk8.6.14 Documentation > Tcl Commands, version 8.6.14 > **after**

Tcl/Tk Applications | Tcl Commands | Tk Commands | [incr Tcl] Package Commands | SQLite3 Package Commands | TDBC Package Commands | tdbc::mysql Package Commands | tdbc::odbc Package Commands | tdbc::postgres Package Commands | tdbc::sqlite3 Package Commands | Thread Package Commands | Tcl C API | Tk C API | [incr Tcl] Package C API | TDBC Package C API

NAME

after — Execute a command after a time delay

SYNOPSIS

after *ms*
after *ms* ?*script script script* ...?
after cancel *id*
after cancel *script script script* ...
after idle ?*script script script* ...?
after info ?*id*?

DESCRIPTION

This command is used to delay execution of the program or to execute a command in background sometime in the future. It has several forms, depending on the first argument to the command:

after *ms*

Ms must be an integer giving a time in milliseconds. A negative number is treated as 0. The command sleeps for *ms* milliseconds and then returns. While the command is sleeping the application does not respond to events.

after *ms* ?*script script script* ...?

In this form the command returns immediately, but it arranges for a Tcl command to be executed *ms* milliseconds later as an event handler. The command will be executed exactly once, at the given time. The delayed command is formed by concatenating all the *script* arguments in the same fashion as the **concat** command. The command will be executed at global level (outside the context of any Tcl procedure). If an error occurs while executing the delayed command then the background error will be reported by the command registered with **interp bgerror**. The **after** command returns an identifier that can be used to cancel the delayed command using **after cancel**. A *ms* value of 0 (or negative) queues the event immediately with priority over other event types (if not installed withn an event proc, which will wait for next round of events).

after cancel *id*

Cancels the execution of a delayed command that was previously scheduled. *Id* indicates which command should be canceled; it must have been the return value from a previous **after** command. If the command given by *id* has already been executed then the **after cancel** command has no effect.

after cancel *script script* ...

This command also cancels the execution of a delayed command. The *script* arguments are concatenated together with space separators (just as in the **concat** command). If there is a pending command that matches the string, it is canceled and will never be executed; if no such command is currently pending then the **after cancel** command has no effect.

after idle *script* ?*script script ...*?

Concatenates the *script* arguments together with space separators (just as in the **concat** command), and arranges for the resulting script to be evaluated later as an idle callback. The script will be run exactly once, the next time the event loop is entered and there are no events to process. The command returns an identifier that can be used to cancel the delayed command using **after cancel**. If an error occurs while executing the script then the background error will be reported by the command registered with **interp bgerror**.

after info ?*id*?

This command returns information about existing event handlers. If no *id* argument is supplied, the command returns a list of the identifiers for all existing event handlers created by the **after** command for this interpreter. If *id* is supplied, it specifies an existing handler; *id* must have been the return value from some previous call to **after** and it must not have triggered yet or been canceled. In this case the command returns a list with two elements. The first element of the list is the script associated with *id*, and the second element is either **idle** or **timer** to indicate what kind of event handler it is.

The **after** *ms* and **after idle** forms of the command assume that the application is event driven: the delayed commands will not be executed unless the application enters the event loop. In applications that are not normally event-driven, such as **tclsh**, the event loop can be entered with the **vwait** and **update** commands.

EXAMPLES

This defines a command to make Tcl do nothing at all for *N* seconds:

```
proc sleep {N} {
    after [expr {int($N * 1000)}]
}
```

This arranges for the command *wake_up* to be run in eight hours (providing the event loop is active at that time):

```
after [expr {1000 * 60 * 60 * 8}] wake_up
```

The following command can be used to do long-running calculations (as represented here by *::my_calc::one_step*, which is assumed to return a boolean indicating whether another step should be performed) in a step-by-step fashion, though the calculation itself needs to be arranged so it can work step-wise. This technique is extra careful to ensure that the event loop is not starved by the rescheduling of processing steps (arranging for the next step to be done using an already-triggered timer event only when the event queue has been drained) and is useful when you want to ensure that a Tk GUI remains responsive during a slow task.

```
proc doOneStep {} {
    if {[::my_calc::one_step]} {
        after idle [list after 0 doOneStep]
    }
}
doOneStep
```

SEE ALSO

concat, **interp**, **update**, **vwait**

KEYWORDS

cancel, delay, idle callback, sleep, time